


Appendix 2

ロー・パワー・デザイン 特別企画 第2弾

# 回路合成ツールによる低消費電力設計

## ―― 電力消費の少ない回路に置き換えたり，回路の一部を止める制御機能を挿入

藤田昌宏

**ここでは論理合成ツールなどが備えている低消費電力化の手法について説明する．こうした処理を，まったくのツール任せにすることはできない．消費電力を削減するために論理合成ツールがどのような回路を生成するのかをある程度推測しながら，利用する必要がある．（編集部）**

現在，おもに利用されているCMOS回路で論理回路を実現する場合，LSI回路中の電力消費として，一般に以下の三つが存在します（**図1**）．

- 回路のスイッチングによる電力消費
- PMOSトランジスタとNMOSトランジスタが一時的に同時にONになったときの貫通電力による電力消費
- 各トランジスタのリーク電流（漏れ電流）から生じる電力消費

LSI回路の設計という観点で考えると，スイッチング電力については，より高位の記述を最適化するほど消費電力の削減効果が大きくなります．半導体技術の微細化とともに，スイッチング電力以外の電力消費も相対的に大きくなってきています．その意味では，上述のすべての電力消費を考慮することがたいせつなのですが，「LSI回路の設計」という立場からは，おもにスイッチング電力を中心に（一部，リーク電流のよる電力消費も考慮しながら）回路最適化を図っていくことになります．

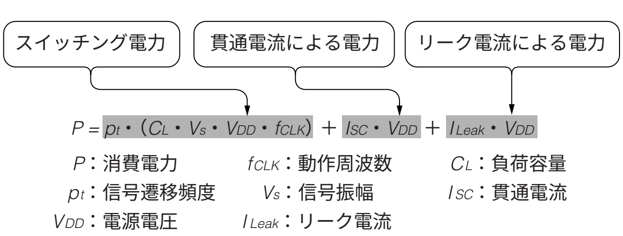



**図1　CMOS回路の消費電力**

現在，おもに利用されているCMOS回路で論理回路を実現する場合，LSI回路中の電力消費として，「スイッチング電力」，「貫通電流による電力」，「リーク電流による電力」の三つが存在する．

システムLSIの設計を**図2**のような流れで考えると，ハードウェア・ソフトウェア分割（アーキテクチャ設計），高位合成（ビヘイビア合成），論理合成，配置・配線の順に消費電力に対する影響は大きくなります．一般に，一つの機能をソフトウェアで実現するかハードウェアで実現するかによって，処理速度はもちろんのこと，消費電力にも大きな差が出ます．また，同じハードウェアで実現するにしても，汎用的なアーキテクチャを採用するか，その設計でのみ適用できる専用のアーキテクチャを採用するかによって，消費する電力が変わります．このような差から見ると，論理合成時の低消費電力化の効果はそれほど大きいとは言えません．

しかしその一方で，どのようなアーキテクチャを採用するにしろ，最終的には論理合成を行います．論理合成処理で得られる消費電力の削減は，アーキテクチャ設計などとは独立した追加の効果と言えます．その意味では非常に重要です．

以上のことを踏まえて，ここでは論理合成，およびそれより少し高位の合成処理における低消費電力化技術について，その概要を紹介します．

### ● 低電力技術の全体像を見通せる技術者が必要

本稿では，おもに以下の手法を中心に説明します．

- 各信号のスイッチング頻度の最小化
  （スイッチング頻度の計算）
- 回路変換，回路最適化による低消費電力化
- 回路動作の停止による低消費電力化
  （ゲーテッド・クロック，演算回路の入力のしゃ断）
- リーク電流の最小化を目ざした回路合成
- 算術式レベルのくふうによる低消費電力化

これらの手法は，それぞれ独立に適用できる場合もありますが，互いに関連していることも多く，一つの最適化手法がほか

**KeyWord**
CMOS，リーク電流，スイッチング電力，ハードウェア・ソフトウェア分割，ビヘイビア合成，論理合成，配置・配線，ゲーテッド・クロック，ファクタリング，ゲート・サイジング

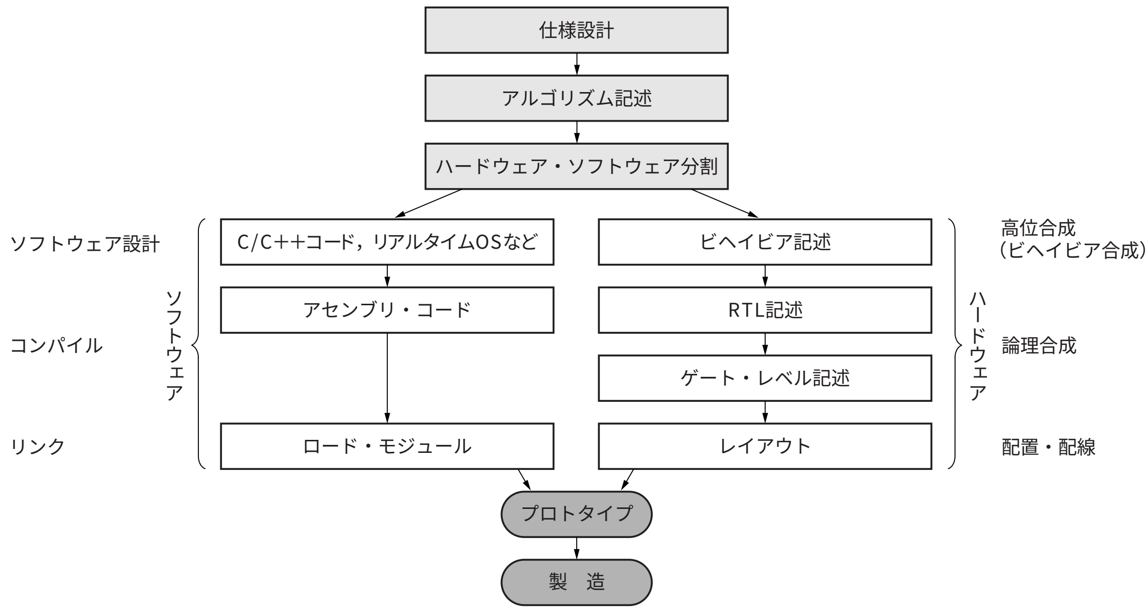



**図2　システムLSI設計の流れ**
**ハードウェア・ソフトウェア分割の工程により，ハードウェア化する機能とソフトウェア化する機能に分かれる．ハードウェア側は，論理合成や配置・配線などを経てLSIになる．ソフトウェア側は，ソフトウェア設計やコンパイルなどを経て，プログラム・コード（バイナリ）になる．**



の最適化手法のじゃまをするというケースも少なくありません．また，EDAツールという観点では，すべての手法が一つのツールに統合されているというわけではありません．複数のツールを連携・利用できる環境をうまく構築する必要があります．

その意味では，低消費電力化技術の全体像を見通せる専門技術者が各種ツールを使い分けられる環境を構築することが重要です．加えて，それぞれの設計者が持つ低消費電力化のためのテクニックやノウハウをツールの中に取り込める環境の構築が強く望まれている分野でもあります．結果的に，EDAベンダから提供されるツールを言われるままに利用している場合と，複数のツール・ベンダから提供されている低消費電力設計用のツールの統合環境を独自に構築している場合で，設計者から見た有効性に大きな差が出てしまっているように見えます．低消費電力技術を広く理解している技術者の育成が急務となっています．

### ● 動的な電力の目安として，スイッチング頻度を求める

**図1**に示すように，CMOS回路の場合，信号値が変化すると，その際に回路中の容量を充放電するような電流が発生し，電力を消費します．したがって，単位時間当たりの回路中の各信号のスイッチング回数を最小化することで，消費電力を低減できます．回路中の各信号のスイッチング頻度を解析する手法として，以下の2種類が考えられます．

a）回路の各入力値がランダムに与えられると考えて，静的に解析する方法

b）特定の回路の入力を設計者が与えて，それに対するスイッチング回数を計測する方法

a）については，組み合わせ回路の場合は比較的単純に計算できます．回路の各入力信号の値は，‘0’，‘1’の値を均等にとるとし，かつランダムに変化するとします．すると，各入力信号が‘0’である確率と‘1’である確率はともに0.5ということになります．

回路中の各ゲートについては，例えば2入力ANDゲートの場合，出力が‘1’となる確率は二つの入力の両方が‘1’となる確率，つまり各入力が‘1’となる確率の積として計算できます．一方，出力が‘0’となる確率は，どちらか少なくとも片方が‘0’となる確率として計算できます．このようにすれば，回路中の各信号が‘1’，‘0’となる確率を計算できます．これを拡張すると，入力値がランダムに変化するという仮定のもとで，回路中の各信号値の変化する確率，つまり一定時間内のスイッチング回数を求めることができます．

AND，OR，EXOR（排他的論理和）の各ゲートに対する計算式を**表1**に示します．$P_A$，$P_B$は二つの入力$A$，$B$が値‘1’になる確率を表しています．例えば，二つの入力が$A$，$B$であるANDゲートの場合，出力値が‘0’から‘1’に変化するのは，少なくとも片方の入力が‘0’であった状態（$1-P_AP_B$．両方とも‘1’ではないということ）から，両方とも‘1’である状態（$P_AP_B$）に変化



**表1　基本ゲートに対する信号遷移確率の計算**

| | $P_{0\to1}$ |
|---|---|
| AND | $(1-P_AP_B)P_AP_B$ |
| OR | $(1-P_A)(1-P_B)(1-(1-P_A)(1-P_B))$ |
| EXOR | $(1-(P_A+P_B-2P_AP_B))(P_A+P_B-2P_AP_B)$ |

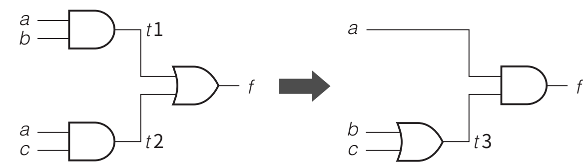



**図3　ファクタリングによる低消費電力化**

共通に利用されている変数をくくり出す操作を行う．これにより回路が簡単になり，結果として回路中の容量が小さくなって消費電力を削減できる．ただし，回路のサイズと信号遷移確率かならずしも比例するわけではないので，注意が必要である．

した場合となり，これらの積として出力信号が‘0’から‘1’に変化する確率を計算できます．なお，この場合，回路中の信号遅延はいっさい存在しないという仮定になっています．

この方法にはいくつかの問題があり，実際の設計ではあまり利用されていません．問題となるのは以下の3点です．

- 順序回路への適用は理論的に可能だが，どれだけ長い時間シーケンスに対して計算しなければならないのかが不明．あるいは，その時間シーケンスが長すぎて，実際に計算が終了しない場合が多い．
- 回路の各入力がランダムに変化するということは実際には起こらない場合が多い．回路の複数の入力値の間には，互いに相関があるのが普通である．
- 上述の計算では，回路中の各ゲートの遅延（入出力伝播遅延時間）をゼロとして計算しているが，実際には遅延を考慮して計算しなければならない．このとき，詳細な回路遅延モデルを使う必要があり，たとえ組み合わせ回路であっても，現実的な時間で計算を完了させることが難しい．

現実的に計算可能な遅延モデルは，すべてのゲートが同じ単一遅延を持つというものに限られます．単一遅延モデルを利用したスイッチング回数の解析は，RTL（register transfer level）の記述に対する消費電力見積もり手法や，論理合成ツールの中のコスト関数の計算に利用されています．

これとは別に，回路中の各ゲートのスイッチング回数を静的に解析するのではなく，シミュレーションなどを行うことで，回路の実際の動作からスイッチング回数を計測し，それにより信号遷移確率を計算する手法も幅広く利用されています．このようにシミュレーションを用いることにより，回路遅延や入力信号の間の相関関係も考慮した解析を行えます．ただし，シミュレーションに要する時間の関係で，計測できるパターンの量はそれほど多くできません．いかに現実の動作に合ったシミュレーション・パターンを用意できるかが重要となります．すでに何度も同じような回路を設計してきている場合は，過去に利用したパターンを一部修正して利用するなど，比較的簡単にシミュレーション・パターンを用意できます．しかし，まったく新規に設計する回路の場合，どのようなシミュレーション・パターンを用意するべきかが不明であり，現実に沿った回路のスイッチング回数を計測できない場合もあります．

### ● 電力消費が少なく，論理的に等価な回路に変換する

回路変換処理や回路最適化処理における低消費電力化手法の基本的な考えかたは，回路中のタイミング的にクリティカルではない箇所（つまり，その箇所の遅延時間が多少延びても，回路全体の信号遅延が大きくならないところ）のゲートや部分回路を，遅延は大きいが消費電力は小さいものに変換することにより，回路全体の低消費電力化を図るというものです．利用できる回路変換や回路合成の手法として，以下のものが挙げられます．

- ファクタリング
- 各種回路構造の変換，バッファの挿入・削除，論理式最適化
- テクノロジ再マッピング
（消費電力を考慮してゲートの割り付けを行う）
- ゲート・サイジング（クリティカル・パス上にないゲートをドライブ能力の小さいものに変更して，消費電力を抑える）
- ゲートのピン割り当ての変更（ゲートの入力ピンから出力ピンまでの遅延が異なることを利用する）

例えばファクタリングの場合，**図3**に示すように共通に利用されている変数をくくり出す操作を行います．これにより回路が簡単になり，結果として回路中の容量が小さくなって消費電力を削減できます．

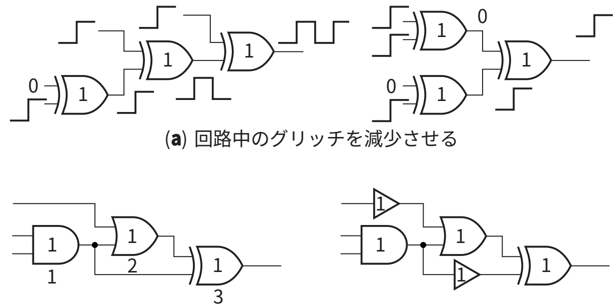
ただし，ファクタリングによって信号の遷移頻度がいつも減少するとは限りません．**図3**の場合，信号$a$が‘1’になる確率が信号$b$や$c$よりも小さいとすると，左側の回路では，中間に現れる信号$t1$，$t2$とも‘1’になる確率はかなり小さくなります（AND ゲートの出力となっているため）．しかし，**図3**の右側の回路では，$t3$は‘1’になる確率の高い信号$b$，$c$のORとなっているため，信号が‘1’になる確率は高いままです．結果として，$t3$の信号遷移確率は，$t1$や$t2$の信号遷移確率よりかなり大きくなってしまいます．

このようにファクタリングなどの論理式レベルの最適化による低消費電力化では，回路のサイズと信号遷移確率はかならずしも比例するわけではないので，注意が必要です．

また，回路構造の変換による低消費電力化の例を**図4**に示します．**図4**(**a**)では回路構造を変換し，入力から出力に至る各信号パスの長さを合わせることで，回路中に不必要に発生するグリッチ(最終的な値に至るまでの過程で信号値が変化する状態)をなくしています．**図4**(**b**)では，同様のことをバッファを適切に導入することで実現しています．このような回路構造の変換による低消費電力化の場合，回路中の遅延をある程度正確に見積もる必要があります．その意味では，レイアウトに依存する遅延(実遅延)などを考慮した解析が重要となります．

このほか，順序回路を対象とする論理合成においては，以下の手法が消費電力削減のために利用されています．

- 状態割り付けを変える
（実験的には，効果が大きくないケースが多い）
- データパス回路のデータのエンコード方式を変える(冗長表現を利用すると，信号の遷移確率を低くできる場合がある)
- リタイミングを行う(フリップフロップをまたいで組み合わせ回路を移動させ，信号パスの遅延の均一化を図る．これにより，回路中のグリッチが減少する)



(**a**) 回路中のグリッチを減少させる

(**b**) バッファを挿入して各信号パスの遅延を合わせる

**図4　回路変換による信号パス間の遅延均一化**

**(a)では回路構造を変換し，入力から出力に至る各信号パスの長さを合わせることで，回路中に不必要に発生するグリッチをなくしている．(b)では，同様のことをバッファを適切に導入することで実現している．**

### ● 回路動作を止めて消費電力を削減する

回路を動作させる必要がないと判断したときに，一時的に動作を止めることで，低消費電力化を図ることができます．

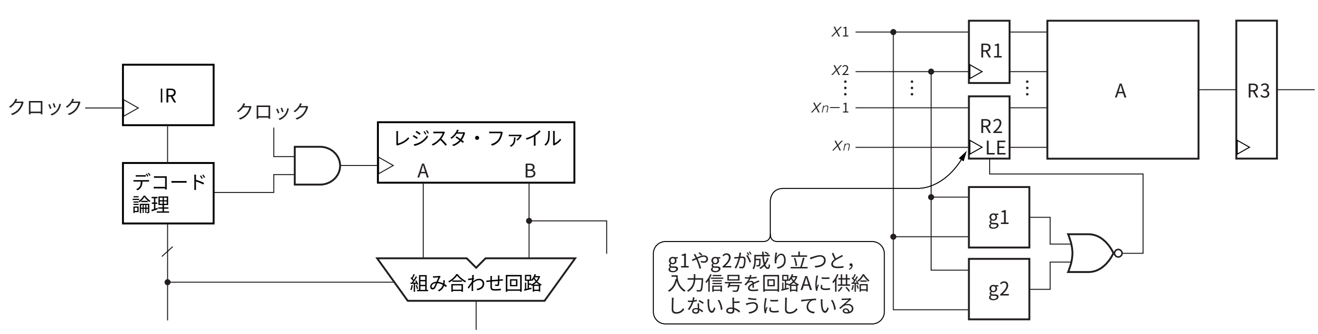
#### 1) ゲーテッド・クロック

ゲーテッド・クロックは，回路を動作させる必要がないと判断したときに，クロックの供給を停止することで，回路を止める手法です(**図5**(**a**))．論理合成ツールなどでも用意されている機能で，手軽に利用できます．ただし，大規模回路ではクロック・スキューが問題となることがあります．クロックを制御する回路の遅延とクロック・スキューがうまく整合しないと正しく動作しないので，回路全体の遅延を考慮して適用する必要があります．レイアウトにもある程度配慮する必要があり，高速回路への適用はそれほど簡単ではありません．

#### 2) 入力のしゃ断による演算回路の低消費電力化

演算回路の場合，利用していない出力であっても，入力信号の値が変化すると自動的に出力が計算され，その分，電力が消費されます．そこで，出力値が不要な場合に，演算回路の入力値を固定することで，演算器の信号遷移が起きないようにする方法が考えられます．**図5**(**b**)に示すように，出力が必要とならない条件をあらかじめ別の回路で計算し，これにより演算器の入力のラッチを固定します．この手法も，ゲーテッド・クロックと同じように遅延の制御が必要なので，注意して利用する必要があります．

Appendix 2




(**a**) ゲーテッド・クロック

(**b**) 演算回路の入力のしゃ断

**図5　回路動作を停止する低消費電力化**

**(a)では，クロックの供給を停止することで，回路を止めている．(b)では，出力が必要とならない条件をあらかじめ別の回路で計算し，これにより演算器の入力のラッチを固定している．**



$P = zu^4 + 4avu^3 + 6bu^2v^2 + 4uv^3w + qv^4$（23個の掛け算）

- Mapleなどのツールによる最適化（Horner形式と呼ばれる）
  $P = zu^4 + (4au^3 + (6bu^2 + (4uw + qv)v)v)v$（17個の掛け算）
- 共通項を抽出した例
  $d_1 = u^2$，$d_2 = v^2$，$d_3 = u \times v$
  $P = d_1{}^2 z + 4ad_1d_3 + 6bd_1d_2 + 4wd_2d_3 + qd_2{}^2$（16個の掛け算）
- さらなる改善例（米国Fujitsu Laboratories of America社のツール）
  $d_1 = v^2$，$d_2 = d_1 \times v$
  $P = u^3(uz + ad_2) + d_1(qd_1 + u(wd_2 + 6bu))$（13個の掛け算）

(**a**) グラフィックス処理で使う算術式

$$\begin{bmatrix} Y_1 \\ X_1 \end{bmatrix} = \begin{bmatrix} 5 & 7 \\ 4 & 12 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

4個の掛け算，2個の加算

→

$D_2 = X_1 + X_2$
$D_1 = D_2 + X_2 \ll 1$
$Y_1 = D_1 + D_2 \ll 2$
$Y_2 = D_1 \ll 1$

3個のシフタ，3個の掛け算

(**b**) 例えば，DCT，IDCT，DFTなどで登場する計算

**図6　算術式の最適化による低消費電力化の例**
**加算やシフトなどで代用することで，掛け算の回数を減らし，低消費電力化を図る．**

### ● タイミングに影響しない箇所のリーク電流を減らす

回路の静的な動作に対する消費電力削減の基本的な考えかたとして，回路中のタイミング的にクリティカルではないパス上の各ゲートに対して，可能な限り下記の方法を適用するというものがあります．

- 電源電圧を下げる
- トランジスタのスレッショルド電圧（しきい値電圧）を上げる
- 可能なら電源配線を切り離す

これらを行うことで，リーク電流など，回路の静的な動作に起因する消費電力を削減できます．

### ● 合成入力の算術式を変形して低消費電力化

論理合成ツールの入力記述を最適化することによる低消費電力化の例として，算術式の最適化による手法を紹介します．もっとも簡単な例は，加算などで代用することで掛け算の回数を減らし，低消費電力化を図る方法です．

**図6**(**a**)を見てください．グラフィックス処理でよく登場する元の式をそのまま計算すると，掛け算が23回現れます．これに対して，通常は，高次の変数を後ろにくくり出すHorner形式と呼ばれる処理を適用します．このようにすると，掛け算の回数を19回に減らせます．

高度なコンパイラなどで行われているように，算術式上の共通項をくくり出して中間変数とし，それを複数回利用して最適化を図ると，掛け算の回数は16回となります．また，最近，開発された高度な方法で共通項を見つけるアルゴリズムを利用すると，掛け算の回数は13回まで減少します．元は23回だったので，半分近くに減少していることになります．

定数との掛け算（一般に，スカラ積と呼ばれる）については，掛け算をシフト演算と加算に分解できます．ある変数を1ビット左（上位のけたの方向）にシフトすると，元の数を2倍したことになり，2ビット分シフトすると4倍したことになります．この性質を利用すると，例えばある変数を5倍したいのであれば，その変数を2ビット分シフトし，元の変数との加算を実行することで実現できます．こうすれば，掛け算器は不要となります．

**図6**(**b**)を見てください．左はDCT（discrete cosine transform；離散コサイン変換）やIDCT（inverse DCT；逆離散コサイン変換）の計算で出てくる算術演算です．これを式のとおりに実行すると，掛け算が4個，加算が2個になります．これに対し，スカラ積をシフト演算と加算に分解し，かつ**図6**(**a**)に出てきたような高度な共通項のくくり出しを行うと，シフトが3個，加算が3個で実現できます．

算術式の最適化については，従来，EDAツールとしてそれほど高度なものは用意されていませんでしたが，最近になって設計者が利用できるレベルのツールも開発されています[注]．今後は，このようなRTLより上位の低消費電力化技術の利用も進んでいくものと考えられます．

注：例えば，米国Fujitsu Laboratories of America社のFarzan Fallah氏がこうしたツールを開発している．このツールについての問い合わせ先はfarzan@us.fujitsu.com.

**ふじた・まさひろ**
**東京大学 大規模集積システム設計教育研究センター（VDEC）教授**

**<筆者プロフィール>**
**藤田昌宏．1985年，東京大学大学院 情報工学専門課程修了．工学博士．同年，富士通研究所入社．論理設計用CAD技術の研究開発に従事．1993年より米国富士通研究所へ出向，CAD研究グループを立ち上げる．2000年3月より東京大学大学院 工学系研究科電子工学専攻 教授．2004年4月より東京大学 大規模集積システム設計教育研究センター（VDEC）教授．ディジタル・システム設計支援技術，とくにシステム・レベル合成，論理合成，論理検証を中心に研究している．**